LINMA23000
HIGH-DIMENSIONAL DATA ANALYSIS AND OPTIMIZATION

Project 1 : Dimensionality reduction methods and nearest neighbor problem

Students :
Alexis AIRSON
Ghali TAHIRI

Professor :
Laurent JACQUES

Teaching Assistants :
Bastien MASSION
Nicolas MIL-HOMENS CAVACO

6 novembre 2024

Table of contents

1 Introduction **2**

2 Identity projection **2**
2.1 Time complexity 2

3 PCA projection **2**
3.1 Time complexity 2
3.2 Study of the error 3

4 Gaussian projection **3**
4.1 Theoretical discussion 3
4.2 Time complexity 4
4.3 Study of the error 4

5 A particular random projection **5**
5.1 Time complexity 5
5.2 Study of the error 5
5.3 Recommendations 6

6 Conclusion **6**

1 Introduction

In many of today's applications, you can think about classification tasks. Just think about self driving cars that must identify traffic signs in real time or face ID recognition on many of your devices. Unfortunately, you can't store a picture of each sign on earth and you don't want your machine to store pictures of you in the morning to state if it's truly you that try to use it. So, we appeal to image processing to accomplish those tasks.

A picture is just a bunch of pixels, each of them containing information about the color displayed. We can thus compress an image in a single vector containing the information of each pixel. But think of it, even low resolution images can represent huge vector. 32 pixels resolution corresponds to $32 \times 32 \times 3 = 3072$ entries. So, if we want to compare new images to the ones from a panel that we already have, the number of comparisons to make will quickly be unreasonable. This is why we appeal to dimensionality reduction to find the nearest neighbor of a new image in a panel that you already have.

In this report, we show the results for different methods used to reduce the dimensionality of the treated data. First, we preprocess the data using the d first singular values of the SVD of the matrix. We then try to use Gaussian distributed matrices (which have nice properties). And finally, we'll use a method relying on d random rows of the DFT matrix. In the following, X denotes the images in which we have to find the nearest neighbors of a set of new images, denoted as Q.

2 Identity projection

2.1 Time complexity

As we iterate through all q in Q, it multiplies the total complexity by $O(N_{query})$, N_{query} being the number of new images in Q. For each vector q, we must compare it to each of the N images in X. This represents d subtractions (as the dimensionality is reduced from n to d before calling this function) followed by a scalar product (d square, $d-1$ additions). Finding the nearest neighbor for every q is then :

$$\min_{i=1,\dots,N} \left\|X_{[:,i]} - q\right\|_2^2 = \min_{i=1,\dots,N} (X_{1,i} - q_1)^2 + \dots + (X_{d,i} - q_d)^2 = \min_{i=1,\dots,N} \sum_{j=1}^{d} (X_{j,i} - q_j)^2 \quad \forall q \text{ in } Q$$

This is thus $N \times (2d + (d-1))$ operations which reduces to $O(N \times d)$. Finding the minimum of those can be done in $O(N)$. The final complexity thus reduces to $O(N_{query} \times N \times d)$ operations. We just have to divide by the FLOPS of the machine to obtain the time complexity. For the identity case, $d = n$.

3 PCA projection

3.1 Time complexity

From documentation, computing U using np.linalg.svd() typically requires $O(min(N,n) \times N \times n)$ operations. The computations of $X' = \Phi X$ and $Q' = \Phi Q$ require, respectively, $O(n \times N \times d)$ and $O(n \times N_{query} \times d)$ operations.
If we want to test multiples values for d, as we don't recompute the svd every time, the expected behavior should be :
$\underbrace{O(n \times N \times d)}_{X'=\Phi X} + \underbrace{O(n \times N_{query} \times d)}_{Q'=\Phi Q} + \underbrace{O(N_{query} \times N \times d)}_{\text{finding nearest neighbor}}.$

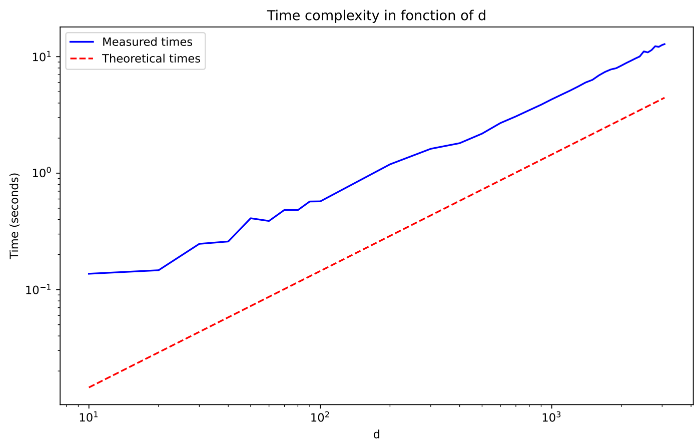

FIGURE 1

Figure1 confirms the theoretical time complexity. In fact, the curve of the measured times follows perfectly the curve of the theoretical time complexity (up to a factor). This factor is probably due to memory access that slows down the process and that is hard to estimate.

3.2 Study of the error

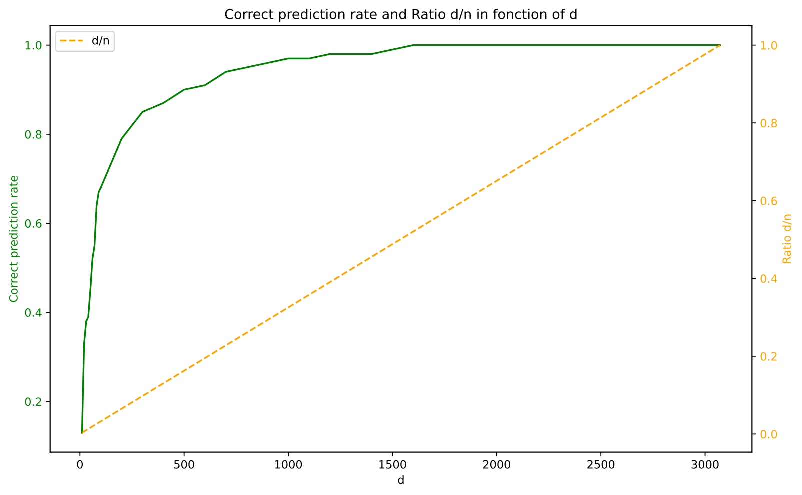

FIGURE 2

We notice in Figure2 that the correct prediction ratio is satisfying even for small values of d, easier to compute, as it follows a logarithmic growth. For example, for $d = 500$, we already have about 90% ratio of correct predictions. It's then very convenient but we need to compute the SVD first. And as it's a deterministic output, using $d = n$ leads to the same results as not projecting at all (100% of correctly predicted ratio).

4 Gaussian projection

4.1 Theoretical discussion

First (Hint 2), we notice that if the entries of Φ are Gaussian distributed, then the entries of $A = \Phi\Psi$ are also Gaussian distributed (note that Ψ is orthonormal). Using this kind of matrices is useful because Gaussian distributed matrices preserve distances in high-dimensional spaces when projecting in lower dimensions. If it is assumed that there exists a basis Ψ in which the images can be represented as sparse vectors, applying Φ preserves the distances, thus the global sparsity of the vector (Hint 1). It even nearly preserves the geometry of sparse vector under projections as those matrices respect the Restricted Isometry Property [1] (RIP) with high probability.

1. https://en.wikipedia.org/wiki/Restricted_isometry_property

4.2 Time complexity

To lower down the effects of randomness, we iterate through 10 runs for every value of d to extract a mean value (this requires yet multiple hours to obtain those results). This appart, we can use the same approach as the one we used for the time complexity of the PCA implementation. The only difference is that we work here with sparse matrices, thus making the computations quicker.
As we don't recompute the wavelet transform for every value of d, We find the same answer as before, i.e $O(n \times N \times d) + O(n \times N_{query} \times d)$ for the scalar products between the matrices and Φ and $O(N_{query} \times N \times d)$ for the nearest neighbor search (note that this is a worst case, in practice, using sparse matrices with an optimized implementation, that we certainly didn't achieve in the notebook, would be more efficient).

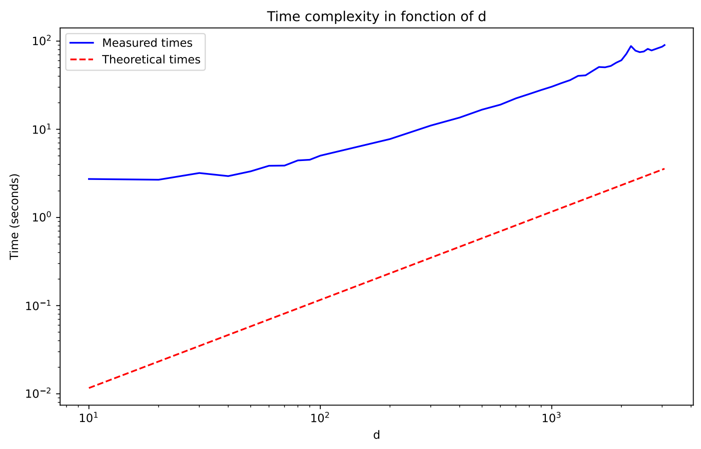

FIGURE 3

Figure3 confirms the theoretical time complexity. In fact, the curve of the measured times follows nearly perfectly the curve of the theoretical time complexity (up to a factor). This factor is probably due to memory access that slows down the process and that is hard to estimate. Moreover, our code is not an optimal implementation.

4.3 Study of the error

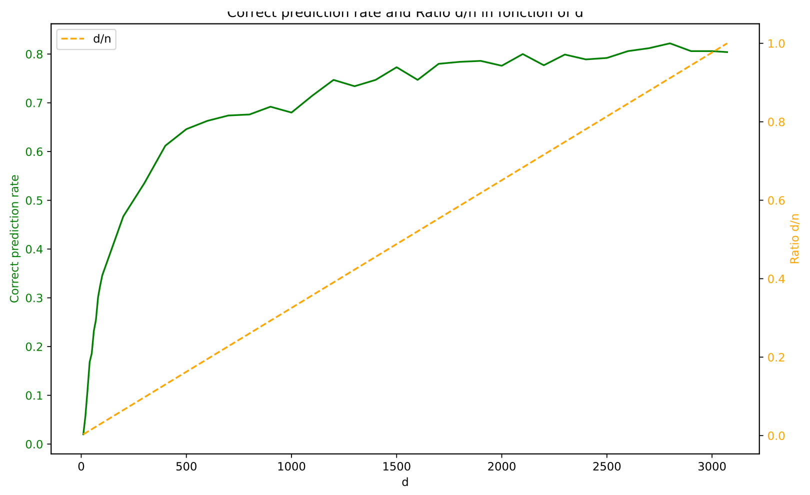

FIGURE 4 – Correct prediction rate of the gaussian projection and Ratio d/n in function of d

Unlike for the PCA, the correct prediction ratio of the Gaussian is less smooth. In fact, we do not reach a 100% ratio even for big values of d and we have to go beyond $d = 1000$ to reach a ratio of 70%. Those fluctuations are for sure due to the randomness in the creation of Φ. Nevertheless, the progression still follows a logarithmic curve.

5 A particular random projection

5.1 Time complexity

The implementation of this part is divided in two. First, we have to go through each column of the matrix Q (N_{query} columns) and the matrix X (N columns). Then, we compute the FFT which time complexity in python is $O(log_2(n))$ and due to the effect of the matrix D, we have to go through all rows of our matrices (n rows) to apply a factor -1 with probability $\frac{1}{d}$. Putting all that together, we find a theoretical time complexity $O(n \times N_{query} \times d \times log_2(n))$.

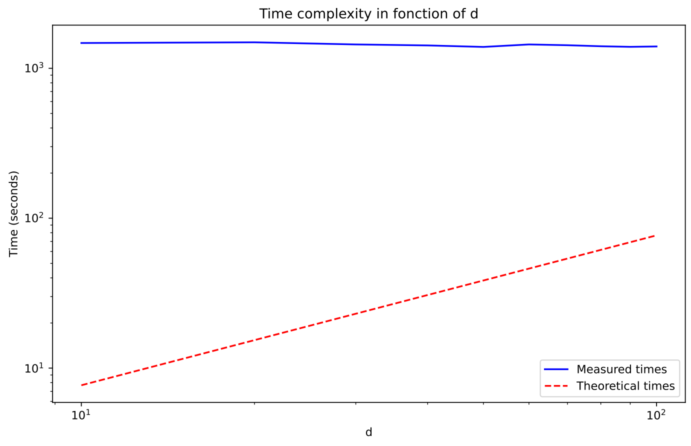

FIGURE 5

However, when we compute our algorithm for several values of d, we notice that the time is quite constant. In general, one iteration takes around 25 minutes whether for $d = 10$ or $d = 100$.

5.2 Study of the error

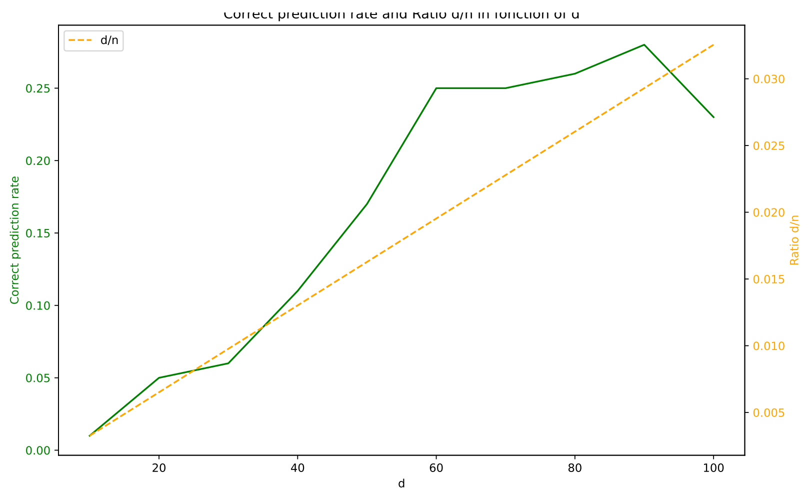

FIGURE 6 – Correct prediction rate of the particular projection and Ratio d/n in function of d

Figure6 shows us how bad this method is. In fact, even if we couldn't try bigger values of d than 100 (due to the long time it takes to compile just one iteration), we can say that the accuracy rate of this projection is not at the same level as the two previous ones. In this sense, for $d = 100$, PCA gave us an accuracy rate above 60% and the Gaussian projection gave a rate around 40%. However, this particular projection gives an accuracy rate between 20% and 25% which is really bad in comparison with the two other ones !

5.3 Recommendations

Projection method	**Time Complexity**
Identity	$O(N_{\text{query}} \times N \times n)$
PCA	$O(n \times N \times d) + O(n \times N_{\text{query}} \times d) + O(N_{\text{query}} \times N \times d)$
Gaussian	$O(n \times N \times d) + O(n \times N_{\text{query}} \times d) + O(N_{\text{query}} \times N \times d)$
Particular random	$O(n \times N_{\text{query}} \times d \times \log_2(n))$

TABLE 1 – Time Complexities for Various Methods

In view of this table and our numerical simulations, we would recommend the PCA projection method and this for two reasons. As seen in Figure1, the time complexity of this method is the best of the three. In fact, even if the theoretical time complexity of the PCA and the Gaussian are similar, in practice, it's not the case. For big values of d, the time complexity of the PCA barely exceeds the order of 10^1 seconds whereas the Gaussian reaches the order of 10^2 seconds if we want an averaged answer to kill effect of randomness (no need to discuss for the particular projection...). The second reason for choosing the PCA is the accuracy. Of the three projection methods, the accuracy of the PCA is the best, even reaching a rate of 100% for higher values of d as seen in Figure2.

6 Conclusion

In this project, we studied the performance of the Projected Nearest Neighbor search across three different projection methods : PCA, Gaussian, and a particular random projection. Our goal was to evaluate these methods in terms of time complexity and accuracy, ultimately to recommend the most efficient approach.

The PCA projection proved to be the best overall method. It offers a manageable time complexity, with theoretical bounds in the order of $O(n \times N \times d) + O(n \times N_{\text{query}} \times d) + O(N_{\text{query}} \times N \times d)$, which aligns closely with observed results. Additionally, PCA achieved the highest accuracy, reaching up to a 100% correct prediction rate for larger values of d. This combination of low computational time and high accuracy makes PCA a highly efficient choice for large-scale nearest neighbor tasks. The measured time complexity, as illustrated in Figure 1, is really acceptable as its of order 10^1 for high values of d. BUT, we need the SVD of the matrix to do this method, which is not always easy to obtain.

The Gaussian projection, although theoretically similar in time complexity to PCA, exhibited higher computational costs in practice. For Gaussian projections, the randomness in matrix generation necessitated averaging over multiple runs to reduce variability, which significantly increased processing time. However, the accuracy of this method was respectable, maintaining a prediction rate close to that of PCA, though requiring larger values of d to reach similar performance levels. Figures 3 and 4 illustrate that while Gaussian projections do provide reasonable results, they are slower and require more computational resources, especially in comparison to PCA.

Finally, the particular random projection performed poorly in both time and accuracy. Its time complexity is constant whatever the value of d but necessitating around 25 minutes for each iteration which is really problematic. The accuracy remained disappointingly low, achieving only around $20\% - 25\%$ prediction accuracy for $d = 100$, far below the rates achieved by PCA and Gaussian projections. This method's inefficiency and inaccuracy make it unsuitable for practical applications.

Overall, based on both theoretical and empirical results, we recommend the PCA projection method for this context due to its balance of time efficiency and high accuracy.